**PE-taak: performance onderzoek**



Groepsleden:

* Emre Demirsatan
* Muhammed Genç

Inhoud

[1. Inleiding 3](#_Toc497909635)

[1.1 Wat is performance onderzoek ? 3](#_Toc497909636)

[1.2 Doel 3](#_Toc497909637)

[1.3 Referentie project 3](#_Toc497909638)

[2. Performance Analysis 4](#_Toc497909639)

[2.1 Metrieke 4](#_Toc497909640)

[2.2 Tools 4](#_Toc497909641)

[3. Oplossingsmethode Game Of Life 5](#_Toc497909642)

[3.1 Referentie Algoritme 5](#_Toc497909643)

[3.2 Performance Optimalisatie 3.3 sdf 5](#_Toc497909644)

[4. Persoonlijke Reflectie 5](#_Toc497909645)

[4.1.1 Emre Demirsatan 5](#_Toc497909646)

[4.1.2 Muhammed Genc 5](#_Toc497909647)

# Inleiding

## Wat is performance onderzoek ?

De letterlijke betekenis van *performance*: ‘het vermogen om veel en/of goed te presteren, prestatie’.

Een performance onderzoek is dus een onderzoek naar het prestatie van het programma. Aan de hand van een performance onderzoek weten de ontwikkelaars wat veel tijd in beslag neemt en/of er nog eventueel wijzigingen kunnen doorgevoerd worden om het programma sneller, beter te laten werken.

Immers is het zo dat een performance onderzoek een programma ontleed en gedetailleerd informatie weergeeft over het geheugengebruik, uitvoeringstijd,...

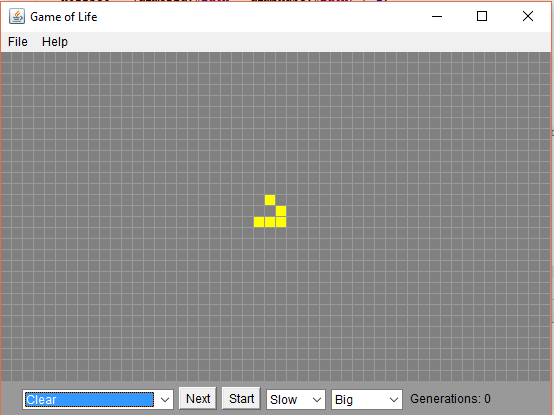
## Doel

Het doel van dit document is het analyseren en verbeteren van source code van Game Of Life. Hierbij gebruiken we een set van leerrijke tools aangeleerd tijdens de les maar ook een deel van tools die zelfstudie van ons vereist.

Dit document zal gedetailleerd de performance oplossing omschrijven die wij toepassen op het project van de sudoku solver. Grafieken zullen de prestaties vergelijken van onze verbeteringen ten opzichte van de originele code. Wij hebben gekozen om Performance analysis toe te passen in de programmeertaal Java.

## Referentie project

Als referentie hebben we gekozen voor de site van Game Of Life. De project die terug te vinden is op deze link: https://bitstorm.org/gameoflife/. De code bevat gedetailleerd omschreven commentaar waardoor het analyseren van de code een stuk gemakkelijker gaat verlopen.



# Performance Analysis

## Metrieke

Als meting voor het bepalen dat een stuk code verbeterd is maken we gebruik van enkele eigenschappen.

* Heap Size
* Snelheid (uitgedrukt in milliseconden)
* CPU cores utilisatie

De verbeterde snelheid wordt weergegeven als een percentage omdat de snelheid voor een groot deel afhankelijk is van de prestatie van uw processor. Hierbij wordt ook aangeduid wat de originele source code was en wat wij gewijzigd hebben om de prestatie te verbeteren.

## Tools

**JConsole**

Komt standaard meegeleverd met de JDK, installatie is niet vereist. JConsole is uitgerust met een dashboard die doormiddel van grafieken de volgende eigenschappen meegeeft:

* Een weergaven over de total heap size ruimte dat er in beslag wordt genomen in het RAM-geheugen.
* Toont het aantal threads die het programma in beslag neemt.
* Een overzicht over hoeveel procent van de capaciteit van de in processor in beslag wordt genomen.

**YourKit**

YourKit biedt ons de mogelijk om te kijken hoe lang het duurt tot de functie gedaan heeft met uitvoeren. Dit is handig om te kijken precies in welke functie versnellingen kunnen worden toegepast. We zullen gebruik maken van de trial version van YourKit samen met IntelliJ.

**JProfiler**

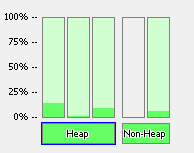
JProfiler toont meer detail over de garbage collection die moet gebeuren. Hierbij toont JProfiler zelfs wat voor soort objecten zich in de Heap bevinden. Dit kan handig zijn om referenties naar een klasse te leggen in plaats van een nieuwe instantie aan te maken. Door middel van het nemen van een snapshot is het ook mogelijk om een algemeen overzicht te krijgen van de Heap.

# Oplossingsmethode Game Of Life

## Referentie Algoritme

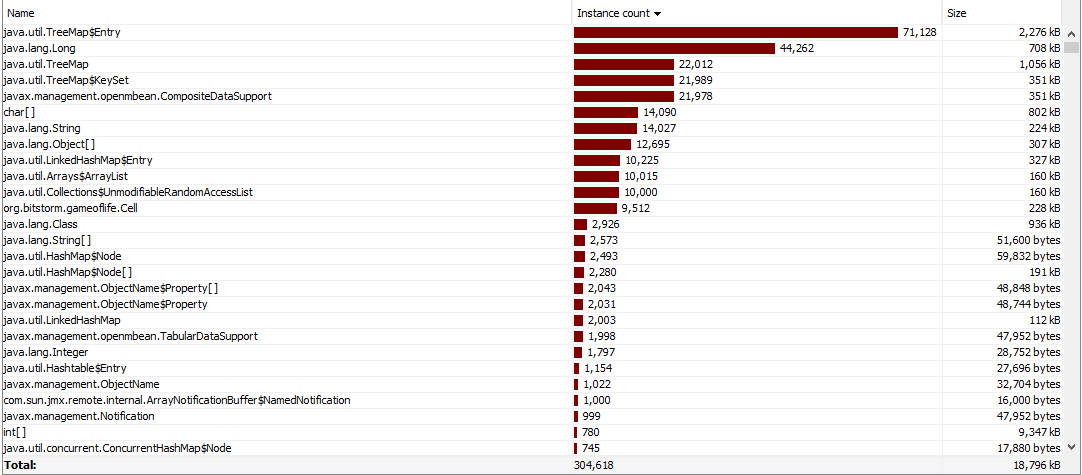
## Performance Optimalisatie

### C:\Users\Muhammed\Documents\School 2017-2018\ProgExpertProject\Knipsel.PNGMemory

De applicatie gebruikt ongeveer 26mb ram, hoe meer cellen je hebt hoe meer ram de applicatie gebruikt. Deze stijging komt voor omdat elk cell een object is op de heap.   
De gebruikte Mb ram blijft constant en dit is een gevolg van de Garbage Collection. Garbage collection gaat random door de heap en verwijdert ongebruikte objecten, hierdoor zal en ram geheugen vrijkomen. Indien en geen Garbage collection is zou de memory blijven stijgen tot er een out of memory exception voorkomt.  
Memory blijft constant dus er is ook geen memory leak in de applicatie.  
  
De Eden space gaat nooit tot 100% want er zijn niet veel objecten die aangemaakt worden en de garbage collection werkt om de zoveel seconden. In de Survivor space zit er zeer weinig objecten. En in de Tenured Gen blijft zeer laag en constant.

De memory is goed genoeg geopstimaliseerd en we hoeven hier niks aan te passen.

### Heap

We gaan dieper in de Heap kijken en zien welk objecten veel worden aangemaakt en hoeveel ze gebruiken.   
 We zien dat er veel objecten zijn van Treemap, Char, string , Long. Omdat er veel instanties betekent het niet dat de size ook groot zal zijn. Als we kijken naar int[] heeft het 780 instanties maar wel 9347kb.

### Final Keyword

# Persoonlijke Reflectie

### Emre Demirsatan

### Muhammed Genc